Optimizations of a Hardware Decoder for Deep Space Optical Communications

Michael K. Cheng, *Member IEEE*, Michael A. Nakashima, Bruce E. Moision, *Member IEEE*, and Jon Hamkins, *Senior Member, IEEE*

***Abstract*— The National Aeronautics and Space Administration (NASA) has developed a capacity approaching modulation and coding scheme that comprises a serial concatenation of an inner accumulate pulse-position modulation and an outer convolutional code (or SCPPM) for deep space optical communications. Decoding of this code uses the turbo-principle. However, due to the non-binary property of SCPPM, a straightforward application of classical turbo decoding is very inefficient. Here, we present various optimizations applicable in hardware implementation of the SCPPM decoder. More specifically, we feature a Super Gamma computation to efficiently handle parallel trellis edges, a pipeline-friendly "maxstar top 2" circuit that reduces the max-only approximation penalty, a low-latency cyclic redundancy check (CRC) circuit for window-based decoders, and a high-speed algorithmic polynomial interleaver that leads to memory savings.**

Using the featured optimizations, we implement a 6.72 megabits-per-second (Mbps) SCPPM decoder on a single field-programmable gate array (FPGA). Compared to the current data rate of 256 Kbps from Mars, the SCPPM coded scheme represents a throughput increase of more than twenty-six fold. Extension to a 50 Mbps decoder on a board with multiple FPGAs follows naturally. We show through hardware simulations that the SCPPM coded system can operate within one dB of the Shannon capacity at nominal operating conditions.

***Index Terms*— Optical communications, turbo decoding, cyclic redundancy check, quadratic polynomial interleaver, FPGA implementation.**

I. Introduction

All of NASA's current deep space missions communicate to Earth using either the radio frequency (RF) spectrum. However the RF spectrum contains much congestion and is susceptible to high diffraction loss due to the spreading of their beam widths. For example, if we use a transmit antenna that is 3.7 meters in diameter (such as one that is mounted on Voyager) and a frequency in X-band to communicate between Earth and Saturn, this transmission beam will spread out to an area over 1000 Earth-diameters wide due to diffraction. We can contrast this result with a system that employs optical communications. If we use a small 10 centimeter optical telescope with wavelength of 1 μm to communicate data between the same Earth-Saturn distance instead, the resulting spot size will only be one Earth-diameter wide. This represents a factor of 1000 concentration of received energy in both horizontal and vertical directions (a factor of 10^6 in power intensity). This improved energy delivery efficiency allows an optical link to operate at a lower transmit power and aperture size while still achieving a higher link data rate. For all of these reasons and more, NASA plans to utilize higher frequency regions in the electromagnetic spectrum to increase the deep space information throughput from 256 Kbps (Mars Exploration Rovers) to tens-of-Mbps and beyond.

Modulation and coding are keys to reliable communications. In the case of an optical link with direct detection, for which we consider, a modulation that has a high peak to average ratio have been shown to be very efficient [1]. Pulse-position modulation (PPM) is one scheme that offers high peak to average power ratio. An M order PPM divides a symbol interval into M possible pulse locations and only a signal pulse is placed into one of these possible positions depending on the information to be transmitted.

Moision and Hamkins compared various concatenated modulation coding schemes with PPM that included Reed-Solomon PPM (RS-PPM), Low-Density-Parity-Check PPM (LDPC-PPM), and convolutional coded PPM. They discovered that a serially concatenated pulse-position modulation (SCPPM) scheme offers the best performance and complexity tradeoff for deep space communications [2].

Modulation is a mapping of bits to symbols transmitted on the channel. This mapping may be considered a code and demodulation as decoding of the code. Conventionally, the modulation and error-correcting code (ECC) are decoded independently, with the demodulator sending its results to the ECC decoder. However, we may consider the combination of the modulation and the ECC as a single large code, which maps user information bits directly to the symbols transmitted on the channel. We could gain several dBs in performance by decoding the ECC and modulation jointly as a single code relative to decoding them independently. An exact maximum-likelihood (ML) decoding of the joint modulation–ECC code would, in most cases of practical interest, be prohibitively complex. However, we may approximate true ML decoding while limiting the decoder complexity by iteratively decoding the modulation and the ECC. This is in fact the "turbo" principle and more details can be found in [3].

Due to the unique structure of SCPPM, a straightforward application of the standard turbo decoding algorithm would be very inefficient. Existing works on turbo optimization, for example that of [4], offer insights but cannot be directly applied to the SCPPM design. Other codes, especially ones designed for the optical channel, that have similar constructions might face the same challenges in their decoding complexity and will benefit from optimizations presented in this work.

This paper is organized as follows: in Section II, we give our

channel assumptions. In Section III we describe the SCPPM code construction and explain why application of classical turbo decoding is not practical. In Section IV, we describe the turbo-like part of the SCPPM decoding.

However, the decoder includes many new techniques that optimize the decoding speed and performance. In Section V we illustrate how to efficiently decode the inner accumulate PPM (APPM) code in SCPPM.

In Section VI we present a hybrid "maxstar top 2" circuit fit for pipelining and achieves a better performance than the max-only approximation with only a small amount additional logic.

In Section VII, we describe the SCPPM interleaver design. The interleaver, characterized by a permutation polynomial, produces a good decoder threshold and a low decoder error floor. The interleaver also has an algorithmic realization that does not require storing the interleaving and deinterleaving mappings, thereby saving memory.

To increase the overall throughput, the SCPPM code trellis can be partitioned into windows and parallel decoders can be applied to the windows. In Section VIII, we provide a low-latency cyclic redundancy check (CRC) circuit that works with window-based decoders.

In Section IX we present various FPGA implementations of the SCPPM decoder that include the featured optimizations and show that SCPPM can operate within one dB of capacity in a nominal deep space mission scenario. We demonstrate that a 6.72 Mbps decoder can be realized on a single FPGA. In addition, we outline a readily achievable path to implementing a SCPPM decoder that can deliver 50 Mbps (enough to transfer compressed high-definition television signals) and beyond for deep space or satellite communications.

II. Channel Assumptions

We consider an optical communications system that uses direct photon detection with a high-order pulse-position modulation (PPM) [5, Chapter 1.2]. An M-order PPM modulation uses a time interval that is divided into M possible pulse locations, but only a single pulse is placed into one of the possible positions. The position of the pulse is determined by the information to be transmitted. A diagram of the optical communications system in discussion is shown in Fig. 1. The information bits $\mathbf{u} = (u_1, u_2, \cdots, u_K)$ are independent identically distributed (i.i.d.) binary random variables assumed to take on the values 0 and 1 with equal probability. The vector $\mathbf{u}$ is encoded to $\mathbf{c} = (c_1, c_2, \cdots, c_n)$, a vector of n PPM symbols. The overall length in bits for a codeword block is $N = n\log_2 M$.

At the receiver, light is focused on a detector that responds to individual photons as illustrated in Fig. 2. For each photon sensed, the detector produces a band-limited waveform for input to the demodulator. This waveform is used to estimate the photon count, k_i, within each slot i. On the Poisson channel, a nonsignaling slot has average photon count n_b and a signaling slot has average count n_s+n_b so that the likelihood

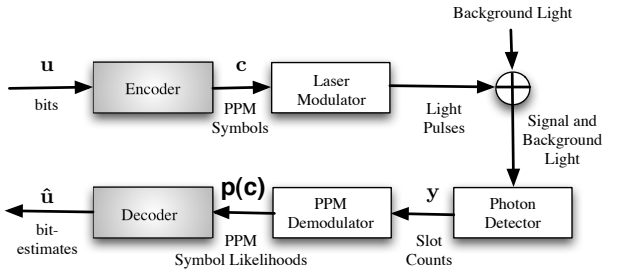

Fig. 1. An optical communications system.

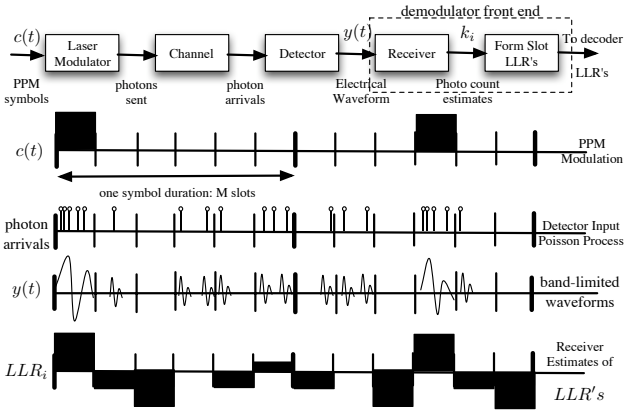

Fig. 2. From PPM symbols to log-likelihood ratios (LLRs).

ratio of slot i is calculated by

$$\begin{aligned} LR(k_i) &= \frac{(n_s+n_b)^{k_i} e^{-(n_s+n_b)}/k_i!}{n_b^{k_i} e^{-n_b}/k_i!} \\ &= e^{-n_s}\left(1+\frac{n_s}{n_b}\right)^{k_i}. \end{aligned} \quad (1)$$

More on the receiver design can be found in [6].

III. The Serially Concatenated Pulse-Position Modulation (SCPPM) Code

The SCPPM encoder, shown in Fig. 3, consists of an outer rate 1/2 constraint length 3 convolutional code, a polynomial interleaver, and an inner accumulate PPM (APPM) code. A block of K information bits $\mathbf{u}$ is cyclic redundancy check (CRC) protected and encoded by the outer convolutional code to yield a length N coded sequence $\mathbf{x}$. This coded sequence is permuted bit-wise to produce the sequence $\mathbf{a}$ that is then filtered by an accumulator and mapped to $n = N/\log_2 M$ PPM symbols $\mathbf{c}$. There are $\log_2 M$ bits per PPM symbol. Due to the APPM bits-to-symbol mapping, the trellis that describes the inner code consists of 2 states and $M/2$ parallel branches between connecting states. We cannot directly apply standard turbo decoding and treat each of the parallel edges separately because doing so would make pipelining difficult and increase decoding latency.

The interleaver and deinterleaver are described by quadratic polynomials and efficient designs are given in Section VII.

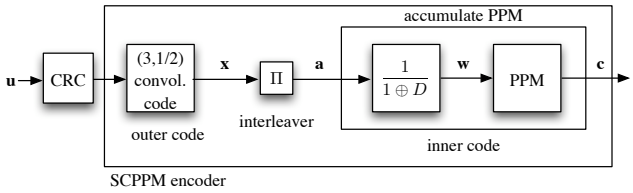

Fig. 3. The SCPPM encoder.

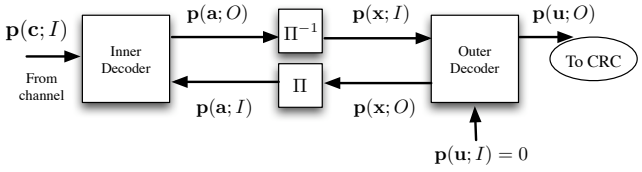

Fig. 4. The SCPPM decoder. Output bits can be directed to a cyclic redundancy check (CRC) to validate codewords.

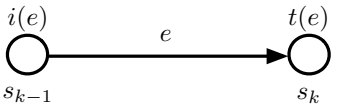

Fig. 5. One stage of a trellis.

IV. SCPPM Decoding

Decoding of the SCPPM code uses the turbo principle. The decoding procedure also incorporates new techniques and components that are not found in the standard turbo approach to optimize hardware implementation. For completeness, we discuss the conventional turbo techniques that are adopted by SCPPM decoding in this Section.

However, efficient SCPPM decoding requires an inner PPM decoder that handles parallel edges in the code trellis, a high-speed algorithmic interleaver (de-interleaver) that works with PPM symbols, and a multiple-bit input cyclic redundancy check circuit (CRC) that works with a windowed decoder. We could not find in existing literature techniques that deal with these topics directly and therefore, we will present our customized approach to each of these subjects in Sections V through VIII.

A high level block diagram of the SCPPM decoder is illustrated in Fig. 4. The symbol I indicates input to the constituent decoders and O indicates output. The inner decoder operates on the modulation code and the outer decoder operates on the convolutional code. Each code is described by a trellis. For each trellis, the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [7] is used to compute the a-posteriori log-likelihood ratios (LLRs) from a-priori LLRs by traversing the trellis in forward and backward directions. Extrinsic information (the difference between the a-posteriori and a-priori LLRs) is exchanged in iteration rather than the a-posteriori LLRs to reduce undesired feedback.

A. Log-Domain Decoding

Each decoder module in the SCPPM decoder applies the BCJR algorithm to the trellis of the constituent code. We use standard notations in the turbo decoding literature [8] and simply restate the calculation of the branch and state metrics inside the inner decoder module. To facilitate hardware realization, the metric computations are done in the log-domain [9], which translates multiplications into additions, and is less sensitive to round-off errors in fixed-point arithmetic.

Let $\mathcal{V}$ be the set of states and $\mathcal{E}$ be the set of directed labeled edges in a trellis. Each edge $e \in \mathcal{E}$ has an initial state $i(e)$ and a terminal state $t(e)$ (see Fig. 5). For each edge e and stage k of the inner code trellis, the BCJR algorithm traverses the trellis in the backward direction to calculate the log branch metric as

$$\bar{\gamma}_k(e) = p_k(\mathbf{a}; I) + p_k(\mathbf{c}; I). \tag{2}$$

The term $p_k(\mathbf{c}; I)$ is the PPM symbol LLR provided by the channel given in (1) and the term $p_k(\mathbf{a}; I)$ is the a-priori symbol LLR provided by the outer decoder. In the same trellis pass, the BCJR algorithm calculates a backward state log metric for each state s and stage k as

$$\bar{\beta}_k(s) = \ln \sum_{e:i(e)=s \in \mathcal{V}} \exp\left(\bar{\beta}_{k+1}(t(e)) + \bar{\gamma}_{k+1}(e)\right). \tag{3}$$

The algorithm then traverses the trellis in the forward direction to calculate the $\bar{\alpha}$'s in the same way. The output LLRs are a function of $\bar{\alpha}$'s, $\bar{\beta}$'s and $\bar{\gamma}$'s. The outer decoder operates on the trellis that describes the outer code using the same principle. This approach is also known as log maximum a-posteriori (log-MAP) decoding [10].

The log sum of exponentials of (3) can be expressed as the max of the exponents plus an adjustment term. This operation is known as the maxstar function:

$$\overset{*}{\max}\ (x, y) \triangleq \ln\left(e^x + e^y\right) = \max(x, y) + \ln\left(1 + e^{-|x-y|}\right). \tag{4}$$

The adjustment term can be precomputed and stored in a lookup table to reduce complexity at an increase in memory usage [11]. We can also ignore the adjustment term entirely to save on memory – this approach is known as max log-MAP decoding. Some of the loss incurred from this approximation can be recovered by scaling the extrinsic information that is passed between the inner and outer decoder [12], [13]. We will introduce a new technique that recovers even more of the loss by adding only a small amount of logic in Section VI.

B. Simplifying Computations with Parallel Trellis Edges

The inner APPM code trellis has 2-states and $2M$ edges per stage as seen in Fig. 6. The forward and backward recursions on this trellis require taking the $\overset{*}{\max}$ of $\frac{M}{2}$ edges per transition between two states. Suppose each 2-input $\overset{*}{\max}$ operation incurs a delay of one clock cycle. A direct implementation of the forward-backward algorithm would require a delay of $\log_2(M/2)$ cycles per transition between two states just for the $\overset{*}{\max}$'s. Barsoum and Moision [2], [14] showed that the computation can be pipelined, reducing the $M/2$-input $\overset{*}{\max}$ operation to a 2-input $\overset{*}{\max}$ operation that is computed in one clock cycle.

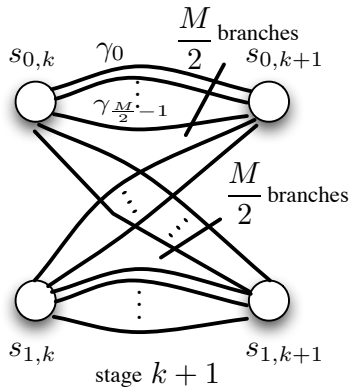

Fig. 6. A single stage of the inner APPM code trellis with $\frac{M}{2}$ parallel edges between connecting states.

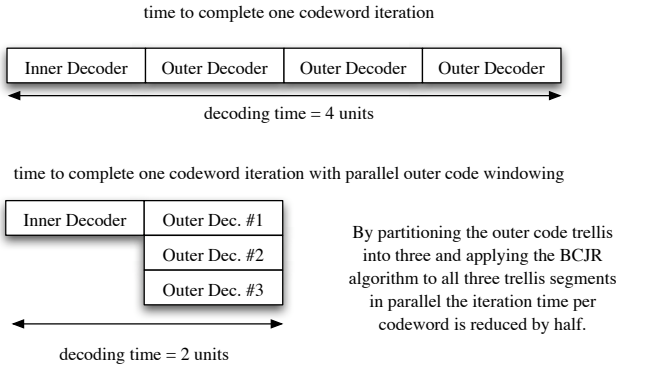

Fig. 7. Windowing increases throughput. For $N = 15120$ and 64 PPM, outer trellis is three times as long as the inner trellis.

Handling of parallel edge transitions in a trellis is not a new notion. However, we are not aware of any existing literature on this topic that would apply directly to our SCPPM code structure. Therefore, in Section V we illustrate how to efficiently deal with the APPM inner code trellis.

C. Fast Modulo Normalization

The BCJR algorithm consists of traversing the code trellis and updating a set of state and branch metrics. Due to the recursive nature of the updates, each of the state metrics in a stage is normalized and clipped by subtracting out the maximum state metric of that stage. Because this update path involves recursion, it cannot be pipelined and becomes the critical path that limits the maximum clock rate at which our design can run on the FPGA. Without normalization, the state metrics can grow unbounded and eventually overflow in a fixed-point hardware implementation. It has been shown for the Viterbi algorithm [15] that as long as the quantization bit width is sufficient to account for the maximum differences between the states metrics, the metrics updates can be allowed to overflow without affecting the result of the computations. This approach naturally extends to the BCJR algorithm and is applied to the SCPPM decoder [11], [16].

D. Partial Statistics

To reduce the channel likelihood storage requirements, we may discard the majority of the channel likelihoods and use partial statistics [17]. This may be accomplished by processing only a subset consisting of the largest likelihoods during each symbol duration–the likelihoods corresponding to the PPM slots with the largest number of observed symbols. The observation of the remaining slots is set to the mean of a noise slot. In low background noise, a small subset may be chosen with negligible loss.

E. A Window Approach to the Outer Decoder

We can partition a code trellis into distinct segments and decode these segments in parallel, therefore increasing the overall throughput.

In iterative decoding, a cyclic redundancy check (CRC) is often used as a stopping rule. While the concept of windowing is not new [18], however, we have not seen in literature a description of an efficient CRC circuit that works with window-based turbo decoders. A conventional CRC is implemented as a linear feedback shift register (LFSR) circuit and takes bit-by-bit input serially. In case of windowing, the decoder outputs multiple bits per clock cycle, many stages apart, and these bits must be buffered into a received word before input into a serial CRC circuit. To remove the serial-input bottleneck and avoid a buffering delay, we provide in Section VIII a CRC circuit that can take as input multiple bits per clock cycle generated by windowing.

The inner SCPPM trellis consists of $n = N/\log_2 M$ symbols or segments. The codeword length N is selected to be 15120 bits and a practical PPM order M is 64. For this setting, the inner trellis will have 2520 segments. The outer trellis is a rate 1/2 code and therefore has $N/2$ or 7560 segments. We can partition the outer code by three and apply window-based BCJR to all three segments in parallel to obtain an overall increase in throughput by a factor of two as seen in Fig. 7.

V. Simplifying Computations in the Inner APPM Decoder

We work with the $\bar{\beta}$ recursion. The $\bar{\alpha}$ computation follow in the same manner. In the product domain, it is straightforward to see an application of the distributive law (multiplication distribution over addition) saves computations on a trellis with parallel edges:

$$\begin{aligned}
\beta_k(s) &= \sum_{e:i(e)=s} \beta_{k+1}(t(e))\gamma_{k+1}(e) \\
&= \sum_{e:i(e)=s,t(e)=s} \beta_{k+1}(s)\gamma_{k+1}(e) \\
&\quad + \sum_{e:i(e)=s,t(e)=\bar{s}} \beta_{k+1}(\bar{s})\gamma_{k+1}(e) \\
&= \left(\beta_{k+1}(s) \sum_{e:i(e)=s,t(e)=s} \gamma_{k+1}(e)\right)
\end{aligned}$$

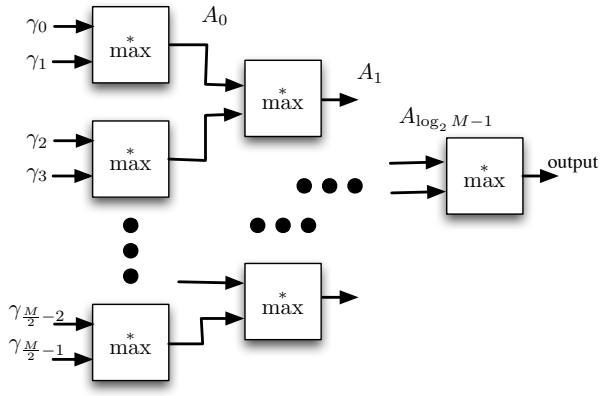

Fig. 8. Pipelined $\overset{*}{\max}$ to compute the Super γ's.

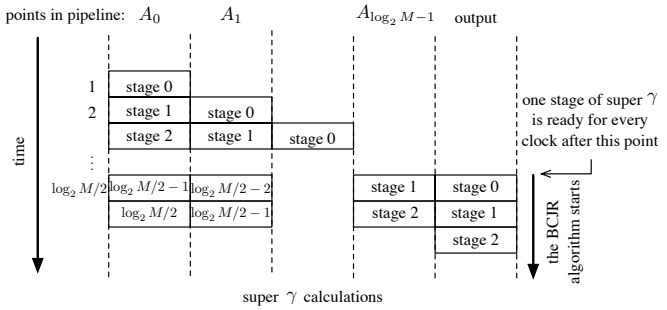

Fig. 9. Scheduling for the Super γ calculations. A stage in the figure represents a stage of γ's in the trellis.

$$+\left(\beta_{k+1}(\bar{s})\sum_{e:i(e)=s,t(e)=\bar{s}}\gamma_{k+1}(e)\right)$$
$$=\quad \beta_{k+1}(s)\gamma'_{k+1}(s,s)+\beta_{k+1}(\bar{s})\gamma'_{k+1}(s,\bar{s}) \quad (5)$$

where $\gamma'_{k+1}(s,s)$, a sum over parallel edges, is referred to as the *Super Gamma* for the state pair (s,s) at stage $k+1$. The same calculation can be made for the state pair $(s,\bar{s})$.

We have an analogous simplification in the log domain via the distributive law (addition distributive over $\overset{*}{\max}$) which can be seen by taking logarithms of both sides of (5)

$$\begin{aligned}\bar{\beta}_k(s) &= \ln\left(\exp\left(\bar{\beta}_{k+1}(s)+\bar{\gamma}'_{k+1}(s,s)\right)\right.\\ &\quad \left.+\exp\left(\bar{\beta}_{k+1}(\bar{s})+\bar{\gamma}'_{k+1}(s,\bar{s})\right)\right)\\ &= \overset{*}{\max}_{\tilde{s}\in\{s,\bar{s}\}}\{\bar{\beta}_{k+1}(\tilde{s})+\bar{\gamma}'_{k+1}(s,\tilde{s})\} \quad (6)\end{aligned}$$

where

$$\bar{\gamma}'_k(s,\tilde{s})=\overset{*}{\max}_{e:i(e)=s,t(e)=\tilde{s}}\{\bar{\gamma}_k(e)\} \quad (7)$$

Since the $\bar{\gamma}'_k$s are not a function of a recursively computed quantity, they may be pre-computed via a pipeline as illustrated in Fig. 8. The schedule for the Super γ calculations is shown in Fig. 9. The pipeline is filled with the first $\log_2 M$ stages of γ's. The decoding BCJR algorithm then starts after the pipeline is filled and thereafter, a set of Super γ values per trellis stage is generated per clock. More on this topic can be found in [2].

VI. Maxstar Only the Top 2 Elements

Implementing the full log-MAP decoder consumes much FPGA resources because each maxstar operation requires a lookup table. For SCPPM, the number of tables required is increased by the potentially high number of parallel edges in the APPM code trellis. To reduce the FPGA resource utilization, we can ignore the adjustment term in the maxstar function and simply use the max operation. However, this simplification, called max log-MAP decoding, comes with a significant decoder performance loss. Wu and Pisuk [12] showed that a confidence factor, which we denote as FF ($0 <$ FF < 1), can be used to weight the extrinsic Log Likelihood Ratios (LLRs) that are passed between two iterative decoders to recover some of the loss incurred from the max log-MAP approach. To recover yet more of the loss, we consider a hybrid approach that takes the maxstar of the top 2 elements in the input array to further reduce the gap between log-MAP and max log-MAP decoding.

A. The Algorithm

Given an array of n elements $\mathbf{x}=(x_1,x_2,\cdots,x_n)$, one method of finding the maxstar of the top 2 elements consists of sorting the array. To do this, one can simply assign two variables top and $top2$ to x_1 and x_2, then compare the rest of the elements in $\mathbf{x}$ with first top and then $top2$. If the compared element is greater than top, we replace $top2$ with top and top with the element. If the compared element is greater than $top2$ only, we replace $top2$ with the element. This procedure is of complexity $O(n)$ and at its completion the two variables will contain the top 2 elements of $\mathbf{x}$ and we perform a $\overset{*}{\max}(top,top2)$ to obtain the desired result. This method would not be efficient to realize in hardware as it requires a state machine, takes n clocks for each array, and cannot be pipelined. We thus develop a "maxstar top 2" algorithm that can be implemented recursively and does not require significant additional circuitry relative to simply taking the max.

The idea is to build from the "max only" pipeline that finds the top element in $\mathbf{x}$. The base case reduces to taking the max of two elements, x_i and x_j, where both $i,j\in\{1,\cdots,n\}$. Instead of propagating only the max of the two elements after each compare, we also feed forward their difference $\Delta_{i,j}=|x_i-x_j|$. In this way, at every stage of the pipeline, we would then be able to maintain not only the current maximum element but also its difference with the next largest element compared so far in the pipe.

Our "maxstar top 2" algorithm takes in 4 inputs and produces two outputs. The inputs are two elements to be compared, x_i and x_j and the difference between each and their next largest element in the previous stage $\Delta_{i,i'}$ and $\Delta_{j,j'}$.

B. The Circuit

The circuit for the two element "maxstar top 2" is given in Fig. 10. The two inputs are denoted here as A and B. Without loss of generality, assume $A>B$. We only need to consider two cases to see how the circuit works. The output of the top multiplexer (mux) will be A and the lower mux will select δ_A.

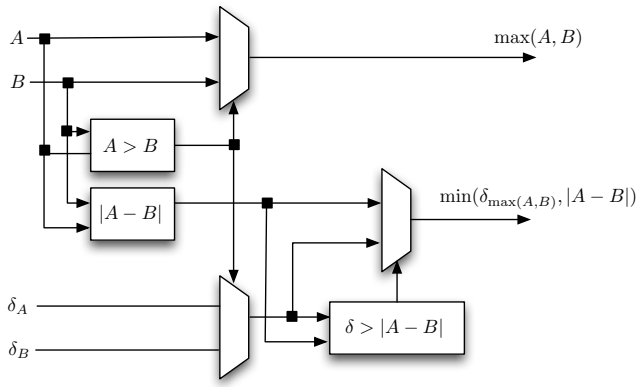

Fig. 10. The maxstar top 2 circuit.

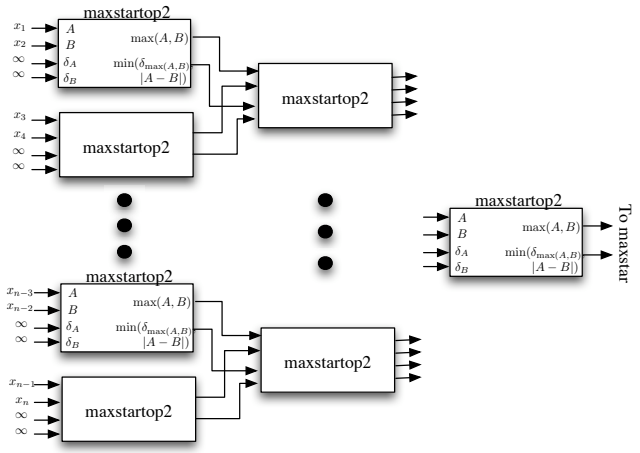

Fig. 11. The maxstar top 2 pipeline.

Case 1: $\delta_A > |A - B|$*:* We state that δ_A is the difference between A and its previous compare, denoted here as A'. Substituting for δ_A we have $|A - A'| > |A - B|$ and we can strip the absolute values because A is the largest of the three so $A - A' > A - B$ and $A' < B$. The element B is closer to A and we output $|A - B|$ in the final mux.

Case 2: $\delta_A \leq |A - B|$*:* We have here $|A - A'| \leq |A - B|$. Stripping the absolute values and rearranging the terms we get $A' \geq B$. The element A' is closer to A than B and we output δ_A in the final mux.

We can inductively see that in the base case with a 2 element input, the "maxstar top 2" circuit outputs the maximum element and the difference between the maximum and the next largest element in previous compares. We can simply replace the maxstar circuit in the pipeline of Fig. 8 with "maxstar top 2" to find the largest element and the difference between the largest and the second largest in all stages before the last. And we use a maxstar circuit in the last stage to calculate the maxstar of the top 2 elements in an array as illustrated in Fig. 11. This approach can be extended to the maxstar of the top 4 elements and so on. The performance of "maxstar top 2" is benchmarked in Section IX.

VII. An Algorithmic Polynomial Interleaver

The interleaver design can affect the decoder threshold and error floor. Choosing a random interleaver permutation will generally lead to a desirable threshold and the key to interleaver design becomes finding a permutation that will also lead to a low error floor. The SCPPM interleaver is characterized by a second order polynomial $f(j) = \kappa j + \ell j^2$. We use choice selections of the parameters κ and ℓ to generate a permutation polynomial that not only exhibits a low error floor but also possesses a simple hardware implementation [19]. Comparison of the SCPPM polynomial interleaver versus the σ-random interleaver [20] is given in Section IX-E.

The interleaver input bit position $f(j) \mod N$ is mapped to output bit position j, i.e.,

$$\mathbf{x}_{f(j)} = \mathbf{a}_j, \quad \mathbf{x}_i = \mathbf{a}_{f^{-1}(i)}.$$

We show that mapping for the $(j+i)$th interleaver position can be expressed as a function of the current interleaver position j:

$$\begin{aligned} [f(j+i)]_N &= \left[\kappa(j+i) + \ell(j+i)^2\right]_N \\ &= \left[\left(\kappa j + \ell j^2\right) + \left(2i\ell j + i(\kappa + \ell i)\right)\right]_N \\ &= [f(j) + g(i,j)]_N \end{aligned} \tag{8}$$

where

$$g(i,j) = 2i\ell j + i(\kappa + \ell i) \tag{9}$$

and $[\cdot]_N$ is the "mod N" operation.

In our design, we assign $N = 15120 = 2^4 \cdot 3^3 \cdot 5 \cdot 7$. Candidate interleavers for this N are of the form $f(j) = \kappa j + 210\lambda j^2$ [19], [21], where λ is a positive integer and κ does not have $2, 3, 5$ or 7 as a factor. Among this class we have observed good performance with the polynomial $f(j) = 11j + 210j^2$. An inverse polynomial is calculated in [2] and given as $f^{-1}(i) = 7331i + 7770i^2$. We use the inverse polynomial to implement the deinterleaver.

A. Interleaver Partitioning for One Clock Read/Write Access

For an M-order PPM modulation, the inner decoder processes a PPM symbol (or $\log_2 M$ bit LLRs) per trellis stage. A straightforward scheduling would be to read one LLR from the interleaver memory per clock. This approach incurs a long latency because the inner decoder would have to wait $\log_2 M$ clocks before proceeding to the next stage. To make interleaving more efficient, we design an approach that allows one clock read/write access. This approach also applies to the deinterleaver.

We illustrate our idea using the $M = 64$ SCPPM decoder with $N = 15120$. The interleaver memory is partitioned into $\log_2 64 = 6$ memory modules. This implementation can be easily adapted for codes with other PPM orders and parameters.

Each module is implemented using Xilinx dual-ported block random access memory (BRAM) as shown in Fig. 12. The input position into the inner decoder j is determined from the output position $f(j)$ of the outer decoder, that is $\text{PaI}[j] \Leftarrow \text{PxO}[[f(j)]_N]$. At each clock, the outer decoder produces two

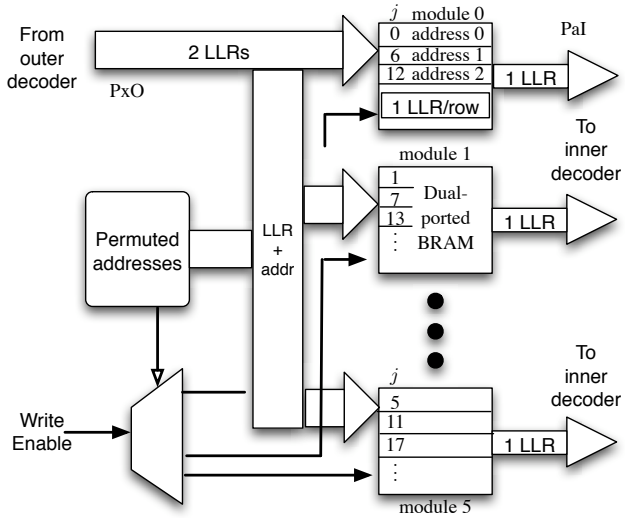

Fig. 12. Interleaver implementation. Permuted addresses can be obtained from a lookup table or computed on-the-fly.

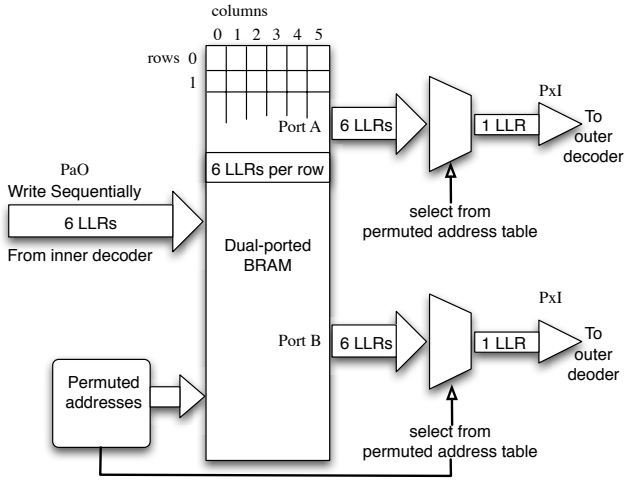

Fig. 13. Deinterleaver implementation. Permuted addresses can be obtained from a lookup table or computed on-the-fly.

LLRs and these are written in permuted order into the BRAMs simultaneously. The address permutation to memory location mapping for the interleaver is given in Table I. The first column consists of the output position $[f(j)]_N$ of the outer decoder in sequential order. The second column consists of the corresponding input position j into the inner decoder. The third column and fourth column are the memory module index (j mod 6) and address ($\lfloor j/6 \rfloor$) in which the corresponding outer decoder output position is stored. The fifth column indicates the trellis stage and the sixth column marks the BCJR window number (for the window-based SCPPM decoder). For example, the 221st LLR, starting from zero, produced by the outer decoder corresponds to the first LLR input for the inner decoder. This LLR is stored in address zero of memory module one. This LLR is calculated at the 110th outer code trellis stage (0-7559) and belongs to the zeroth window segment (out of three).

The outer decoder writes to the interleaver BRAMs in permuted order using the mapping of Table I. As we march down the table entries, we see that there will be no write conflicts at any time because the period of memory module writes is six and only two LLRs are produced by the outer decoder each clock. During interleaver reads, the inner decoder accesses the BRAM entries in sequential order. That is, at the first clock, the inner decoder reads the first entry (address 0) of each of the six memory modules and increases the address pointers by one. The six LLRs read correspond to PaI[0] through PaI[5] and are highlighted by bold face fonts in Table I. At the next clock, the inner decoder reads the second entry (address 1) of each memory module and again updates the address pointer. These six LLRs read correspond to PaI[6] through PaI[11] and so on.

The deinterleaver is implemented as one big chunk of memory as illustrated in Fig. 13. The output LLRs generated by the inner decoder is written sequentially six at a time into "one row" of the dual-ported BRAM. The outer decoder then reads the LLRs in permuted order two at a time from the deinterleaver. The address permutation table for the deinterleaver is the same as that of the interleaver given in Table I, with the exception that the header corresponds to that of the second row. For example, PxI[862], the 862nd LLR (starting from zero) input to the outer decoder should be read from the second column zeroth row of the deinterleaver BRAM. The control logic reads the desired two rows and then selects the correct entry out of each row. One can see from the table that there are no read conflicts.

With the above interleaver and deinterleaver design, the LLRs produced or required by a stage of trellis decoding can be written to or read from memory in one clock cycle.

B. Removing the Need to Store Interleaver Mappings

We can avoid the need to store Table I in memory by computing the memory module and address for a specific interleaved position on-the-fly. The interleaver is partitioned into $C = \log_2 M$ distinct memory blocks each with $n = \frac{N}{C}$ entries for fast read and write access. Each interleaver position $[f(j)]_N$, for $j = [0, 1, \cdots, N-1]$, is mapped to a corresponding index pair $\left(r_{f(j)}, q_{f(j)}\right)$ where $r_{f(j)} \triangleq [f(j)]_C$ is the index into one of the C memory modules and $q_{f(j)} = \left[\left\lfloor \frac{f(j)}{C} \right\rfloor\right]_n$ is the index into one of the n address entries in each module. Each stage of the inner APPM decoding produces C LLRs. Because the modulo and division operations are costly to implement in hardware, we describe a procedure that calculates the interleaver indexing pair for the set of C LLRs desired by the current stage based on the set of C indexing pairs computed in the previous stage. We begin with a proposition.

Proposition 1: If $C \mid N$, then $[[f(j)]_N]_C$is equivalent to $[f(j)]_C$

Proof: A nonnegative number f modulo C can be obtained by continuously subtracting C from f until f becomes less than C. If $C \mid N$, the number f modulo N can be obtained by subtracting $n = \frac{N}{C}$ multiples of C from f. Therefore, $[[f(j)]_N]_C$is equivalent to $[f(j)]_C$. ■

$\mathbf{x}_{f(j)}$: $f(j)$	$\mathbf{a}_j$: j	module	addr	stage	window
$\mathbf{x}_i$: i	$\mathbf{a}_{f^{-1}(i)}$: $f^{-1}(i)$	column	row	stage	window
0	**0**	**0**	**0**	**0**	**0**
1	15101	5	2516	0	0
2	382	4	63	1	0
3	1203	3	200	1	0
4	2444	2	407	2	0
5	4105	1	684	2	0
⋮	⋮	⋮	⋮	⋮	⋮
221	**1**	**1**	**0**	**110**	**0**
⋮	⋮	⋮	⋮	⋮	⋮
862	**2**	**2**	**0**	**431**	**0**
⋮	⋮	⋮	⋮	⋮	⋮
5040	10080	0	1680	2520	1
5041	10061	5	1676	2520	1
⋮	⋮	⋮	⋮	⋮	⋮
10080	5040	0	840	5040	2
10081	5021	5	836	5040	2
⋮	⋮	⋮	⋮	⋮	⋮
15119	439	1	73	7559	2

TABLE I

ADDRESS PERMUTATION TABLE FOR THE INTERLEAVER (TOP HEADER) AND DEINTERLEAVER (BOTTOM HEADER).

1) Step 1: Initialization: We assign the constant $\Delta_q = [2C\ell]_n$ and set the initial modulus values using (8) an (9) to

$$r_{f(0)} = [f(0)]_C, \cdots, r_{f(C-1)} = [f(C-1)]_C, \quad (10)$$

$$r_{g(0)} = [g(C,0)]_C, \cdots, r_{g(C-1)} = [g(C,C-1)]_C, \quad (11)$$

as well as the initial quotient values

$$q_{f(0)} = \left[\left\lfloor \frac{f(0)}{C} \right\rfloor\right]_n, \cdots, q_{f(C-1)} = \left[\left\lfloor \frac{f(C-1)}{C} \right\rfloor\right]_n, \quad (12)$$

$$q_{g(0)} = \left[\left\lfloor \frac{g(C,0)}{C} \right\rfloor\right]_n, \cdots, q_{g(C-1)} = \left[\left\lfloor \frac{g(C,C-1)}{C} \right\rfloor\right]_n. \quad (13)$$

2) Step 2: Loop for $stage = 1 : n-1$ and $i = 0 : C-1$: Note that each i update is implemented by an individual circuit. Therefore, we have C circuits working in parallel, each calculating the interleaver indexing pair for each of the C LLRs needed by decoding of the inner code. First, we expand

$$\begin{aligned} g(C, i+C) &= 2c\ell(i+C) + C(\kappa + \ell C) \\ &= g(C,i) + 2C^2\ell \end{aligned} \quad (14)$$

and define $g(l) \triangleq g(C,l)$. Using this definition, (8), and (14) together, we update the modulus as

$$\begin{aligned} r_{f(stage\cdot C+i)} &= [f((stage-1)\cdot C + i + C)]_C \\ &= [f((stage-1)\cdot C + i) \\ &\quad + g((stage-1)\cdot C + i)]_C \\ &= r_{f((stage-1)C+i)} \end{aligned} \quad (15)$$

because for all stages

$$\begin{aligned} r_{g(stage\cdot C+i)} &= \left[g((stage-1)\cdot C + i) + 2C^2\ell\right]_C \\ &= 0. \end{aligned} \quad (16)$$

We express the functions

$$f((stage-1)\cdot C + i) = q_f C + r_f \quad (17)$$

and

$$g((stage-1)\cdot C + i) = q_g C + r_g. \quad (18)$$

We follow by updating the quotient for f as

$$\begin{aligned} q_{f(stage\cdot C+i)} &= \left[\left\lfloor \frac{q_f C + r_f + q_g C + r_g}{C} \right\rfloor\right]_n \\ &= \left[q_f + q_g + \left\lfloor \frac{r_f + r_g}{C} \right\rfloor\right]_n \\ &= \left[q_{f((stage-1)C+i)} + q_{g((stage-1)C+i)}\right]_n \end{aligned} \quad (19)$$

because from (15) $r_f < C$ and from (16) $r_g = 0$. We can then update the quotient for g as

$$\begin{aligned} q_{g(stage\cdot C+i)} &= \left[\left\lfloor \frac{g((stage-1)\cdot C + i) + 2C^2\ell}{C} \right\rfloor\right]_n \\ &= \left[\left\lfloor \frac{q_g C + r_g}{C} \right\rfloor + 2C\ell\right]_n \\ &= \left[q_{g((stage-1)\cdot C+i)} + \Delta_q\right]_n. \end{aligned} \quad (20)$$

The memory module for the interleaver position $f(stage\cdot C + i)$ is then $r_{f(stage\cdot C+i)}$ and the address entry is $q_{f(stage\cdot C+i)}$.

C. Circuit Description for the Algorithmic Interleaver

The derivations of equations (15) and (16) indicate that the memory module index for each LLR and each stage stays the same throughout the trellis. Plugging in parameters to the initial values of (10) will show that the memory module indices per trellis stage has a period C. This observation is confirmed by Table I. In the table, the memory module indices take on

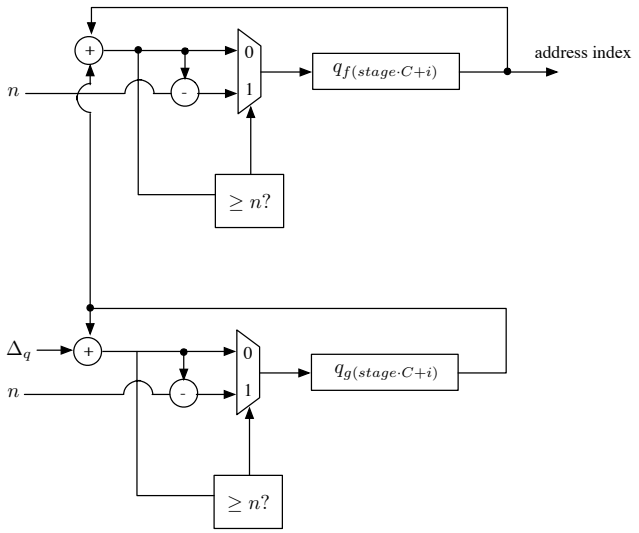

Fig. 14. A circuit that computes the address into the memory module for the ith LLR in each stage, where $i = [0, \cdots, C-1]$. There will be C instances of this circuit one for each LLR.

the values $[0, C-1, C-2, \cdots, 2, 1]$ for every stage and this pattern repeats for all stages. Consequently, we only need to calculate the address entry for each LLR in each stage. A circuit that implements equations (19) and (20) to compute the address entry per LLR per trellis stage is given in Fig. 14. Note that there will be C such circuits one for each LLR per stage working in parallel.

This algorithmic interleaver removes the need to store Table I. Implementation only requires a small number of gates. This memory saving benefit is even more evident in multiple decoder instantiations on one FPGA because many copies of the same Table need not be stored. We characterize logic and memory tradeoff number for our SCPPM decoder in Section IX-A.

VIII. A Low-Latency CRC

A straightforward hardware implementation of a cyclic redundancy check (CRC) is a linear feedback shift register (LFSR). A block of information bits with the associated CRC check bits are shifted into the LFSR circuit one bit at a time. After the entire block is input to the circuit, the state of the registers indicates whether the CRC is verified. A CRC can be used together with iterative turbo decoding to flag codeword errors or to stop decoding iterations.

To increase the throughput of turbo decoding, the code trellis can be partitioned into distinct windows and multiple decoders can be applied to these windowed trellis segments in parallel. Doing so will generate multiple decoded bits per clock and these bits must be buffered into a received word before running the serial CRC circuit. To avoid this buffering delay, we propose a low-latency CRC circuit that can handle multiple-bit inputs per clock.

A. Polynomial Description of CRCs

Let us write a length k binary message block $\mathbf{m} = (m_{k-1}, m_{k-2}, \cdots, m_0)$, that is to be protected by a CRC, in polynomial form:

$$m(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + \cdots + m_0 \tag{21}$$

Let the length n CRC protected codeword be $\mathbf{c} = (c_{n-1}, c_{n-2}, \cdots, c_0)$ or

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_0 \tag{22}$$

and the CRC generator be

$$g(x) = g_{n-k}x^{n-k} + \cdots + g_0. \tag{23}$$

The CRC polynomial $r(x)$ is calculated by first shifting the message polynomial left by $n-k$ positions and then by taking the modulo $g(x)$ operation

$$r(x) = R_{g(x)}\left[m(x) \cdot x^{n-k}\right], \tag{24}$$

where $\deg[r(x)] < n-k$. The codeword block can also be written as

$$\begin{aligned} \mathbf{c} &= [\mathbf{m}, 0, \cdots, 0] \oplus \mathbf{r} \\ &= \begin{bmatrix} m_{k-1}, \cdots, m_0, & \underbrace{0, \cdots, 0}_{n-k} \end{bmatrix} \\ &\quad \oplus [r_{n-k-1}, \cdots, r_0] \end{aligned} \tag{25}$$

($\oplus$ is the binary XOR operations) or

$$c(x) = m(x) \cdot x^{n-k} + r(x). \tag{26}$$

To verify the CRC of a codeword block $\hat{c}(x) = c(x) + e(x)$ that may be corrupted by an error polynomial $e(x)$, we calculate

$$\begin{aligned} R_{g(x)}[\hat{c}(x)] &= R_{g(x)}\left[m(x) \cdot x^{n-k} + r(x) + e(x)\right] \\ &= R_{g(x)}\left[R_{g(x)}\left[m(x) \cdot x^{n-k}\right] + R_{g(x)}[r(x)]\right. \\ &\quad \left. + R_{g(x)}[e(x)]\right] \\ &= r(x) + r(x) + R_{g(x)}[e(x)] \\ &= R_{g(x)}[e(x)]. \end{aligned} \tag{27}$$

Therefore, if the remainder is zero, the CRC passes and the error polynomial is zero. If the remainder is nonzero, then the codeword is corrupted. Note that we won't be able to construct the error polynomial $e(x)$ from the CRC remainder $R_{g(x)}[e(x)]$.

B. Hardware Description of CRC checks

A CRC is simply a modulo operation and can be implemented by a linear feedback shift register (LFSR) for dividing polynomials. The circuit for multiplying by a polynomial $h(x)$ and dividing by a polynomial $g(x)$, each with degree up to ℓ, is given in Fig. 15. For division only, simply set $h(x) = 1$ ($h_0 = 1$, every other coefficients to 0). After the entire codeword is shifted into the circuit, the quotient of the division operation is given by the bits that are shifted out and the remainder is given by the register state. More information on LFSRs can be found in [22, Linear Switching Circuits].

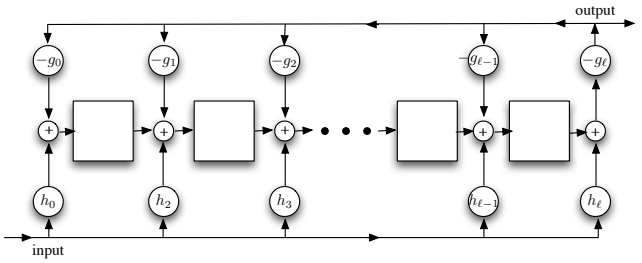

Fig. 15. A circuit for multiplying by $h(x)$ and dividing by $g(x)$.

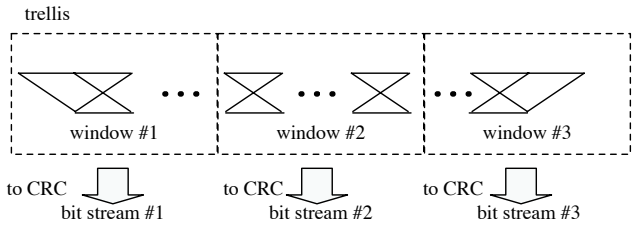

Fig. 16. A trellis windowed by three leads to three simultaneous decoded bit streams.

C. CRC Circuit for Windowed-Based Turbo Decoding

In windowed-based turbo decoding, the output bit streams to be fed into the CRC are generated in parallel as seen in Fig. 16. We describe how a CRC circuit can be modified to handle this parallelism. Let the code trellis be partitioned into j distinct windows. The codeword polynomial can be written as

$$c(x) = c_1(x)\, x^{s_1} + c_2(x)\, x^{s_2} + \cdots + c_j(x). \tag{28}$$

We can then write the check polynomial as

$$\begin{aligned} R_{g(x)}[c(x)] &= R_{g(x)}[c_1(x)\, x^{s_1} + c_2(x)\, x^{s_2} + \cdots + c_j(x)] \\ &= R_{g(x)}\left[R_{g(x)}[c_1(x)\, x^{s_1}] + R_{g(x)}[c_2(x)\, x^{s_2}]\right. \\ &\quad \left. + \cdots + R_{g(x)}[c_j(x)]\right] \\ &= R_{g(x)}\left[R_{g(x)}[c_1(x)\,\kappa_1(x)]\right. \\ &\quad \left. + R_{g(x)}[c_2(x)\,\kappa_2(x)] + \cdots + R_{g(x)}[c_j(x)]\right], \end{aligned} \tag{29}$$

where $\kappa_i = R_{g(x)}[x^{s_i}]$, $i = 1, 2, \cdots, j-1$, and each $\kappa_i(x)$ can be pre-calculated. The CRC LFSR circuit for the window-based decoder will consist of both feed-forward and feedback tap connections. The feed-forward taps are given by the XOR of $\kappa_i(x)$'s and the feedback taps are given by the generator $g(x)$.

D. A CRC Circuit for the Window-Based SCPPM Decoder

A practical realization of the SCPPM code scheme is to use PPM order 64. The SCPPM decoder in our implementation is windowed by three, as detailed in Section IV-E, to double the overall throughput. With the outer code trellis having 7560 stages, each windowed by three segment has 2520 stages. We use a 22-bit CRC with generator $g(x) = x^{22}+x^5+x^4+x^3+1$ to check the output of the windowed SCPPM decoder. The CRC indicates whether a correct codeword decision is reached and can be used to stop the iteration process. Using techniques presented in this section, we precompute the polynomials $\kappa_1(x) = R_{g(x)}\left[x^{5040}\right]$ and $\kappa_2(x) = R_{g(x)}\left[x^{2520}\right]$ and generate three circuits (shown in Fig. 17) to check the output bit stream of each window. We can optimize and consolidate the three circuits into one by XORing the three output bit streams according to the feed-forward taps before input to the CRC circuit.

IX. Decoder Results

The SCPPM decoder for PPM order $M = 64$ is currently implemented on a Xilinx Virtex II-8000 FPGA part, speed grade 4 (XC2V8000-4), which sits on a Nallatech BenDATA-WS board. The memory requirement is reduced by taking only the top 8 channel LLRs as decoder input. The channel LLRs input to the decoder are quantized to 8-bits, 5 for dynamic range and 3 for precision.

We have implemented three versions of the decoder. The first is the log-MAP decoder with normalization and clipping circuits for the state metrics. The backward recursion state metrics $\bar{\beta}$'s are clipped to 8 bits before being stored into RAMs. The forward recursion state metrics $\bar{\alpha}$'s are calculated as needed and not stored. The remaining variables in the data path are allowed to grow and not stored.

The second is the max log-MAP decoder with modulo normalization. The $\bar{\beta}$'s are allowed to grow in dynamic range up to 16-bits (plus a 3-bit precision for a total of 19-bits) before being stored into RAMs. Again, the $\bar{\alpha}$'s are calculated as needed and not stored. All other metrics are allowed to grow in width and not stored.

The third is the window-based max log-MAP decoder. The outer code trellis is partitioned into three.

We only had the opportunity to complete place and route for a fourth variation of the decoder, the "maxstar top 2" implementation, and did not get a chance to finish the wrapper around the decoder. But we did produce a bit-exact software of the "maxstar top 2" decoder and used it to generate accurate simulation results.

A. Resource Utilization

The FPGA utilization report for all decoder implementations is given in Table II. Each decoder implementation has two rows of utilization numbers in percentages. The first row reports the slices (or logic) utilization. The second row reports the BRAM utilization. Each utilization is then further broken down into inner (decoder), outer (decoder), and miscellaneous components. The sum of the components equals to the total. Only percentages are given and the actual number of logic slices and BRAM blocks can be computed from the Xilinx Virtex II-8000 specification. For this part, the FPGA has 46592 logic slices and 168 BRAM blocks. Miscellaneous blocks are modules external to the decoder that consume resources such as circuitries and buffers for the FPGA interface. The channel symbol memory and state metric storage are all implemented using Xilinx internal dual-ported block RAMs (BRAMs) for all decoders.

The $\overset{*}{\max}$ lookup tables (LUTs) for the log-MAP decoder are realized as simple multiplexers with hard coded inputs

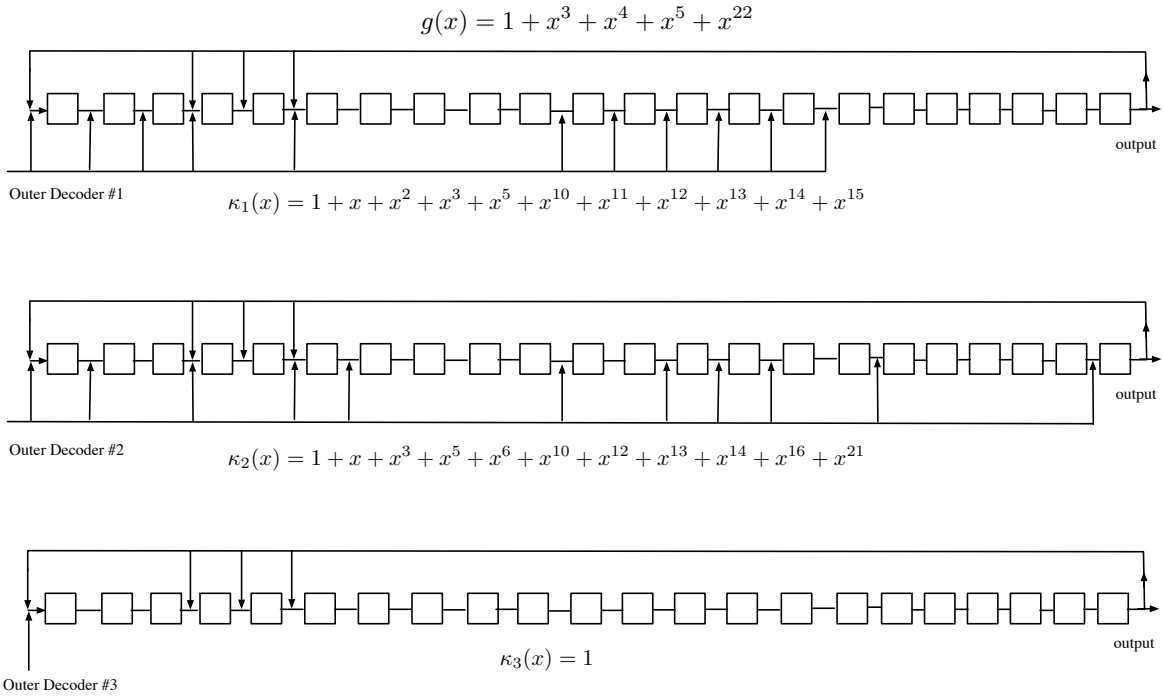

Fig. 17. CRC circuits for the polynomial $g(x) = x^{22} + x^5 + x^4 + x^3 + 1$ and the SCPPM windowed by three decoder.

that function like read-only memories (ROMs). The log-MAP decoder also allocated logic to normalization and clipping. We can tradeoff resource and performance by removing the LUTs and the clipping circuits. The max log-MAP decoder avoids LUTs and uses modulo normalization to reduce the logic utilization from 64% to 42% at a performance cost of 0.2 dB. Moreover, we can compromise between resource and performance by using the hybrid maxstar top 2 log-MAP decoder. The logic utilization for this approach is 54%, just half way between the 64% of log-MAP and 42% of max log-MAP. The performance cost is also half way in between at 0.1 dB.

In the log-MAP decoder we store the interleaver mappings of Table I as LUTs on BRAMs. For the max log-MAP version, we avoid storing the interleaver mapping in BRAM and use the algorithmic approach. This allows us to compare the memory savings in going to an algorithmic interleaver. The synthesis tool reports a savings of 453 Kb in BRAM memory which is 15% of the 3.02 Mb total on the Virtex-II 8000. The cost in logic to implement the algorithmic interleaver is reported as only 40 out of 46592 slices.

We also provide the report for window-based max log-MAP decoder. As expected, windowing the inner code trellis by three and operating three parallel outer decoder increases the outer decoder logic percentage from 5% to 15%. However, this implementation increases the throughput by a factor of two. The maximum clock rate and throughput based on 7 average iterations for all decoder designs are given in Table III.

B. Error Rate Performance

The decoder performance is shown in Fig. 18. The frame loss rate (FLR) is plotted versus n_s the average signal photons per pulse slot in dB. Each frame is a codeword of $k = 7560$ information bits. A frame loss is declared when the decoder decision could not converge to the correct codeword in the maximum number of allowed iterations which is set at 32. Out of the 7560 bits, 2 bits are used to terminate the trellis and 22 bits are used for CRC. The CRC polynomial is $x^{22} + x^5 + x^4 + x^3 + 1$ and has an undetected word error probability of approximately $7 \cdot 2^{-22} = 1.67 \times 10^{-6}$ assuming 7 average iterations. To reduce the overall undetected word error rate, the decoder runs a minimum number of iterations first before validating the CRC. In doing so, the undetected word error probability is lowered to roughly the product of the frame loss rate and 1.67×10^{-6}, a very small value.

We make the following observations in the performance plot. Fixed-point implementation (circle-line) has a 0.1 dB loss compared to the floating-point decoder (dashed-line). Clipping and and normalization of the state metrics led to a floor at 10^{-5}. Max log-MAP decoder (square-line) has a 0.6 dB loss compared to log-MAP decoding (circle-line). Max log-MAP decoder with a scaling of the extrinsic information by 0.5 (diamond-line) recovers 0.4 dB out of the 0.6 dB lost. Note that only the extrinsic information at the output of the inner decoder is scaled by a factor between 0 and 1. The extrinsic information at the output of the outer decoder is untouched. The clipping and normalization floor is lowered by using modulo arithmetic.

Also notice that using the "maxstar top 2" (triangle-line) circuit in the inner decoder, the log-MAP outer decoder, and a scaling of inner decoder extrinsic information by 0.625, we are able to recover another 0.075 dB in signal energy. A 0.5 scale factor can be implemented in hardware by simply a right shift by 1. A scaling of 0.625 is the sum of a right shift by 1

Decoder Type	Percent **Slices** Used / Percent **BRAM** Used	Percentage Breakdowns for **Inner**	**Outer**	**Miscellaneous**
log-MAP	64%	52%	6%	6%
	60%	19%	9%	32%
max log-MAP	42%	31%	5%	6%
	51%	19%	9%	23%
window-based max log-MAP	53%	32%	15%	6%
	80%	19%	30%	31%
hybrid maxstar top 2 log-MAP	54%	43%	5%	6%
	51%	19%	9%	23%

TABLE II

COMPARING THE DIFFERENT SCPPM DECODER IMPLEMENTATIONS ON THE VIRTEX-II 8000 FPGA. THIS FPGA HAS A TOTAL OF 46592 LOGIC SLICES AND 168 BRAM BLOCKS.

	log-MAP	**max log-MAP**	**window-based max log-MAP**	**hybrid maxstar top 2 log-MAP**
Maximum Clock	23 MHz	63 MHz	63 MHz	63 MHz
Throughput	1.23 Mbps	3.36 Mbps	6.72 Mbps	3.36 Mbps

TABLE III

MAXIMUM CLOCK RATE AND THROUGHPUT FOR THE DIFFERENT SCPPM DECODER DESIGNS ON THE XILINX VIRTEX-II 8000 FGA.

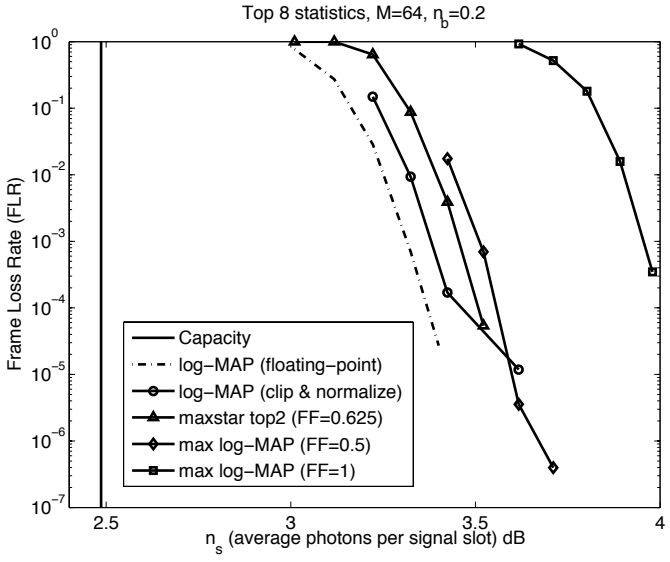

Fig. 18. SCPPM decoder performance on the Poisson channel under a nominal deep-space mission scenario.

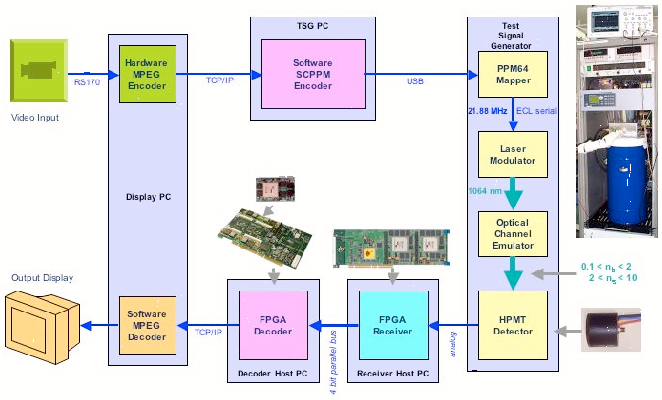

Fig. 19. An end-to-end SCPPM optical link demonstration.

and right shift by 3.

C. End-to-End Demonstration

We have successfully demonstrated [23] an end-to-end SCPPM optical communications system as shown in Fig. 19. We are able to deliver quality MPEG-2 video from a camera to a display using this setup. The transmitter employs a 1064 nm wavelength (Nd: YAG) solid state laser to modulate a stream of SCPPM encoded symbols. The PPM pulses are then sent over a fiber optic channel. At the receiving end, a Hybrid Photo-Multiplier Tube (HPMT) photon counting detector is used and the receiver assembly converts the photon counts into LLRs for our FPGA decoder. The results of the experimental runs at various operating points are plotted in Fig. 20. There are two experimental runs, one at 4 Mbps and the other at 6 Mbps both use only the top 8 statistics and a maximum of 7 iterations. These two curves are compared to a curve generated by using a software simulated Poisson channel and the stand-alone FPGA decoder. We see that the experimental curves match very closely to the stand-alone FPGA result. The end-to-end performs within 1.5 dB of channel capacity. At a frame loss rate of 10^{-5} the number of signal photons per pulse slot is 2.67 and this corresponds to 3/2.67=1.12 information bits per photon.

D. Performance Comparison vs. Reed-Solomon-PPM scheme

A legacy ECC used in many previous and current NASA missions is the Reed-Solomon code. In Fig. 21 we compare the Reed-Solomon PPM (RS-PPM) coded scheme versus the SCPPM coded scheme and show that in a nominal mission scenario, SCPPM out performs RS-PPM by 3 dB. The results are generated using software simulation. To match the code rates we use the SCPPM code parameters $(N, K) = (16398, 8199)$ and the RS-PPM code parameters $(4085, 2047)$. We choose 64

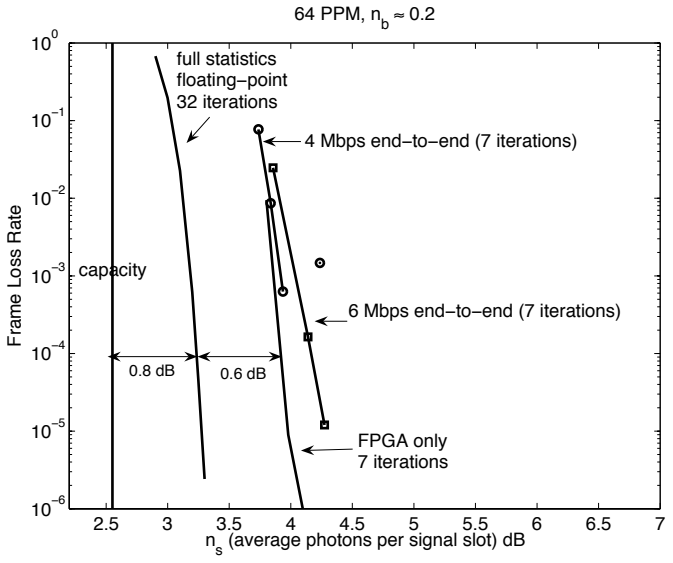

Fig. 20. End-to-end system performance. A frame is equivalent to a codeword of 7560 information bits.

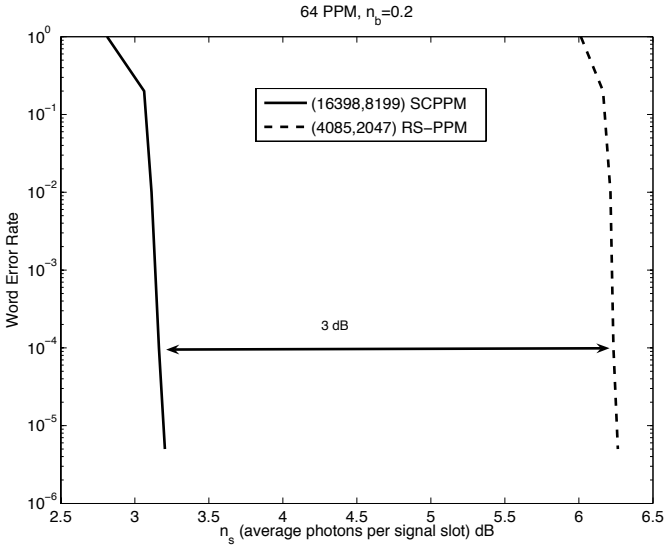

Fig. 21. Comparison of SCPPM versus RS-PPM under a nominal deep space mission scenario.

PPM as a reasonable order and an average background count of 0.2 photons per slot.

E. Performance Comparison of Different Interleavers

The interleaver design affects the error rate performance and error floor of SCPPM. Through simulation, we show that the SCPPM algorithmic polynomial interleaver performs as well as a σ-random interleaver and has no observable error floor. The error rate curves are plotted in Fig. 22. We see that for both the word error rate (WER) and bit error rate (BER) the two interleavers produced almost identical decoder performance. For deep space missions, where minimum WER floor requirements are generally that of 10^{-4}, the two interleavers meet the specifications.

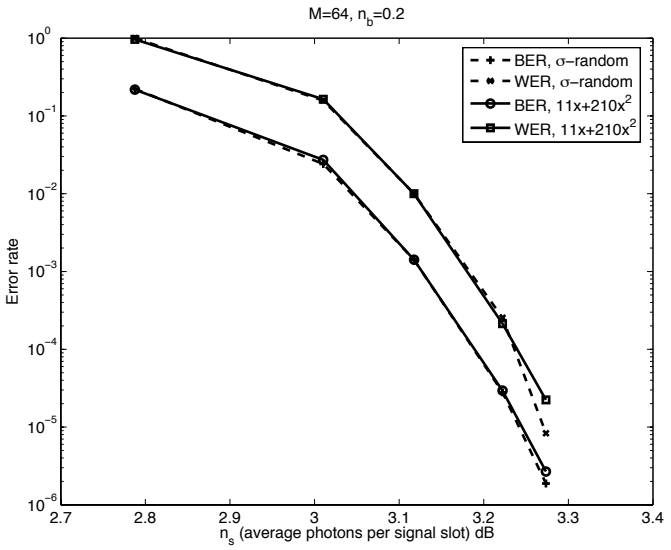

Fig. 22. Comparison of the SCPPM polynomial interleaver versus the σ-random interleaver.

F. Path to 50 Mbps and Beyond

We achieved a 6.72 Mbps decoder on a single Xilinx Virtex-II FPGA. Currently, Xilinx has available the Virtex-II Pro FPGA part that is manufactured with a smaller micron-process and features more BRAMs. We have completed a place and route of our fastest design on the Virtex-II Pro. Results indicate that the SCPPM decoder can deliver 8 Mbps at 7 average iterations. We can add another stage of parallelism to our design so that the inner decoder and outer decoder can work on two codewords simultaneously and are not idle at any time. Doing so doubles our throughput to 16 Mbps per FPGA. Moreover, we can realize multiple instances of our decoder on the Nallatech BenNuey-4E PCI board that has slots for three daughters each capable of hosting two Virtex-II Pro FPGAs. This migration path leads to a 96 Mbps SCPPM decoder that is fit for deep space optical communications. We can further increase the throughput to hundreds of mega-bits and beyond by implementing a lower order SCPPM decoder, such as 16-PPM, for terrestrial applications where a smaller PPM order actually achieves higher capacity due to the shorter distance between the transmitter and receiver.

X. Conclusion

The serially concatenated pulse-position modulation (SCPPM) capacity approaching code is designed by NASA to support deep space optical communications at mega-bits-per-second (Mbps) and beyond. The non-binary property of SCPPM makes direct application of conventional turbo decoding very inefficient. In this work, we introduced new techniques that optimize the overall decoder throughput and performance: a simplified Super Gamma to handle the unique inner accumulate PPM code structure, a pipeline friendly "maxstar top 2" circuit that reduces the max-only

approximation penalty, a low-latency CRC circuit that works with window-based decoders, and a rapid access algorithmic interleaver that can compute the permutation mappings on-the-fly.

To convey the efficacy of our techniques, we implemented three variations of the SCPPM decoder on a Xilinx Virtex-II 8000 FPGA and summarized their tradeoffs. Through hardware simulation we demonstrated that a single instance of our SCPPM decoder can generate information bits at more than 6.72 Mbps and that the code design can perform within one dB of capacity under nominal mission conditions. We believe that our hardware optimizations are applicable to other non-binary modulation and code schemes that are characterized by a high peak to average power ratio designed to fit the requirements of long distance optical communications.

ACKNOWLEDGMENTS

The authors would like to thank Bill Farr, Jonathan Gin, Kevin Quirk, and David Zhu at JPL for providing insightful technical discussions and for putting together the end-to-end laser communications demonstration.

REFERENCES

[1] J. R. Pierce, "Optical channels: practical limits with photon counting," *IEEE Trans. Commun.*, vol. 26, pp. 1819–1821, Dec. 1978.

[2] B. Moision and J. Hamkins, "Coded modulation for the deep space optical channel: serially concatenated PPM," *JPL Interplanetary Network Progress Report*, vol. 42-161T, May 2005.

[3] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," *The Telecom. and Data Acquisition Progr. Rep.*, vol. 42, pp. 1–20, Nov. 1996.

[4] Z. Wang, Y. Tang, and Y. Wang, "Low hardware complexity parallel turbo decoder architecture," in *IEEE Proc. Int. Symp. Circuits and Systems*, May 2003.

[5] H. Hemmati, *Deep Space Optical Communications*. John Wiley & Sons Inc., 2006. ISBN 0-470-04002-5.

[6] K. J. Quirk and L. B. Milstein, "Optical PPM with sample decision photon counting," in *Proc. IEEE Global Telecom. Conf.*, (St Louis, MO, USA), IEEE, Dec. 2005.

[7] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, pp. 284–287, March 1974.

[8] W. E. Ryan, "A turbo code tutorial." Available on-line at http://www.ece.arizona.edu/~ryan/publications/turbo2c.pdf, 1997.

[9] T. V. Souvignier, *Turbo Decoding for Partial Response Channels*. PhD thesis, University of California, San Diego, 1999.

[10] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 260–264, Feb. 1998.

[11] G. Montorsi and S. Benedetto, "Design of fixed point iterative decoders for concatenated codes with interleavers," *IEEE J. Select. Areas Commun.*, vol. 19, pp. 871–882, May 2001.

[12] P. H. Wu and S. M. Pisuk, "Implementation of a low complexity, low power, integer-based turbo decoder," in *Proc. IEEE Global Telecom. Conf.*, vol. 2, (San Antonio, Texas), pp. 946–951, IEEE, 2001.

[13] J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *Electronic Letters*, vol. 36, pp. 1937–1939, Nov. 2000.

[14] M. Barsoum and B. Moision, "Method and apparatus for fast digital turbo decoding for trellises with parallel edges." JPL Novel Tech. Rep., July 2004. No. 4123.

[15] A. P. Hekstra, "An alternative to metric rescaling in Viterbi-decoders," *IEEE Trans. Commun.*, vol. 37, pp. 1220–1222, Nov. 1989.

[16] M. K. Cheng, B. E. Moision, J. Hamkins, and M. A. Nakashima, "Implementation of a coded modulation for deep space optical communications," in *UCSD first workshop on Information Theory and its applications*, (San Diego, CA), Univ. Calif. San Diego, Feb. 2006.

[17] B. Moision and J. Hamkins, "Reduced complexity decoding of coded-pulse modulation using partial statistics," *JPL Interplanetary Network Progress Report*, vol. 42-161, May 2005.

[18] R. Akella and J. K. Wolf, "On the parallel MAP algorithm," in *IEEE fourth workshop on multimedia signal processing*, pp. 371–376, IEEE, Oct. 2001.

[19] J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. Inform. Theory*, vol. 51, pp. 101–119, Jan. 2005.

[20] D. Divsalar, S. Dolinar, F. Pollara, and R. J. McEliece, "Weight distributions for turbo codes using random and nonrandom permutations," *The Telecom. and Data Acquisition Progr. Rep.*, vol. 42, pp. 56–65, Aug. 1995.

[21] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Trans. Inform. Theory*, vol. 52, pp. 1249–1253, March 2006.

[22] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*. Cambridge, MA, USA: The M.I.T. Press, 1961.

[23] A. Biswas and et. al., "Palomar receive terminal for the Mars Laser Communications Demonstration project," *IEEE Special Issue on Tech. Adv. in Deep Space Comm. and Tracking*, Nov. 2006. Submitted.